ID2S Password-Authenticated Key Exchange Protocols

Xun Yi, Fang-Yu Rao, Zahir Tari, Feng Hao, Elisa Bertino, Ibrahim Khalil and Albert Y. Zomaya

Abstract—In two-server password-authenticated key exchange (PAKE) protocol, a client splits its password and stores two shares of its password in the two servers, respectively, and the two servers then cooperate to authenticate the client without knowing the password of the client. In case one server is compromised by an adversary, the password of the client is required to remain secure. In this paper, we present two compilers that transform any two-party PAKE protocol to a two-server PAKE protocol on the basis of the identity-based cryptography, called ID2S PAKE protocol. By the compilers, we can construct ID2S PAKE protocols which achieve implicit authentication. As long as the underlying two-party PAKE protocol and identity-based encryption or signature scheme have provable security without random oracles, the ID2S PAKE protocols constructed by the compilers can be proven to be secure without random oracles. Compared with the Katz et al.'s two-server PAKE protocol with provable security without random oracles, our ID2S PAKE protocol can save from 22% to 66% of computation in each server.

Index Terms—Password-authenticated key exchange, identity-based encryption and signature, Diffie-Hellman key exchange, decisional Diffie-Hellman problem

1 Introduction

To secure communications between two parties, an authenticated encryption key is required to agree on in advance. So far, two models have existed for authenticated key exchange. One model assumes that two parties already share some cryptographically-strong information: either a secret key which can be used for encryption/authentication of messages, or a public key which can be used for encryption/signing of messages. These keys are random and hard to remember. In practice, a user often keeps his keys in a personal device protected by a password/PIN. Another model assumes that users, without help of personal devices, are only capable of storing "human-memorable" passwords.

Bellovin and Merritt [4] were the first to introduce password-based authenticated key exchange (PAKE), where two parties, based only on their knowledge of a password, establish a cryptographic key by exchange of messages. A PAKE protocol has to be immune to on-line and off-line dictionary attacks. In an off-line dictionary attack, an adversary exhaustively tries all possible passwords in a dictionary in order to determine the password of the client on the basis of the exchanged messages. In on-line dictionary attack, an adversary simply attempts to login repeatedly, trying each possible password. By cryptographic means only, none of PAKE protocols can prevent on-line dictionary attacks. But on-line attacks can be stopped simply by setting a threshold to the number of login failures.

- *Xun Yi, Zahir Tari and Ibrahim Khalil are with the School of Computer Science and IT, RMIT University, Melbourne, Australia.*
- *Fang-Yu Rao and Elisa Bertino are with the Department of Computer Science, Purdue University, USA.*
- *Feng Hao is with the School of Computing Science, Newcastle University, UK.*
- *Albert Y. Zomaya is with the School of Information Technologies, University of Sydney, Australia.*

Since Bellovin and Merritt [4] introduced the idea of PAKE, numerous PAKE protocols have been proposed. In general, there exist two kinds of PAKE settings, one assumes that the password of the client is stored in a single server and another assumes that the password of the client is distributed in multiple servers.

PAKE protocols in the single-server setting can be classified into three categories as follows.

Password-only PAKE: Typical examples are the "encrypted key exchange" (EKE) protocols given by Bellovin and Merritt [4], where two parties, who share a password, exchange messages encrypted by the password, and establish a common secret key. The formal model of security for PAKE was firstly given in [3], [8]. Based on the security model, PAKE protocols [1], [2], [5], [10], [11], [16], [20], [22] have been proposed and proved to be secure.

PKI-based PAKE: PKI-based PAKE protocol was first given by Gong et al. [17], where the client stores the server's public key in addition to share a password with the server. Halevi and Krawczyk [18] were the first to provide formal definitions and rigorous proofs of security for PKI-based PAKE.

ID-based PAKE: ID-based PAKE protocols were proposed by Yi et al. [32], [33], where the client needs to remember a password in addition to the identity of the server, whereas the server keeps the password in addition to a private key related to its identity. ID-based PAKE can be thought as a trade-off between password-only and PKI-based PAKE.

In the single-server setting, all the passwords necessary to authenticate clients are stored in a single server. If the server is compromised, due to, for example, hacking or even insider attacks, passwords stored in the server are all disclosed. This is also true to Kerberos [12], where a user authenticates against the authentication server with his username and password and obtains a token to authenticate against the service server.

To address this problem, the multi-server setting for PAKE was first suggested in [15], [19], where the password of the client is distributed in n servers. PAKE protocols in the multi-server setting can be classified into two categories as follows.

Threshold PAKE: The first PKI-based threshold PAKE protocol was given by Ford and Kaliski [15], where n severs, sharing the password of the client, cooperate to authenticate the client and establish independent session keys with the client. As long as $n-1$ or fewer servers are compromised, their protocol remains secure. Jablon [19] gave a protocol with similar functionality in the password-only setting. MacKenzie et al. proposed a PKI-based threshold PAKE protocol which requires only t out of n servers to cooperate in order to authenticate the client. Their protocol remains secure as long as $t-1$ or fewer servers are compromised. Di Raimondo and Gennaro [26] suggested a password-only threshold PAKE protocol which requires fewer than $1/3$ of the servers to be compromised.

Two-server PAKE: Two-server PKI-based PAKE was first given by Brainard [9], where two servers cooperate to authenticate the client and the password remains secure if one server is compromised. A variant of the protocol was later proved to be secure in [27]. A two-server password-only PAKE protocol was given by Katz et al. [23], in which two servers symmetrically contribute to the authentication of the client. The protocol in the server side can run in parallel. Efficient protocols [21], [29], [30], [31] were later proposed, where the front-end server authenticates the client with the help of the back-end server and only the front-end server establishes a session key with the client. These protocols are asymmetric in the server side and have to run in sequence. Yi et al. gave a symmetric solution [34] which is even more efficient than asymmetric protocols [21], [29], [30], [31]. Recently, Yi et al. constructed an ID2S PAKE protocol with the identity-based encryption scheme (IBE) [35].

In this paper, we will consider the two-server setting for PAKE only. In two-server PAKE, a client splits its password and stores two shares of its password in the two servers, respectively, and the two servers then cooperate to authenticate the client without knowing the password of the client. Even if one server is compromised, the attacker is still unable to pretend any client to authenticate against another server.

A typical example is the two-server PAKE protocol given by Katz et al. [23], which is built upon the two-party PAKE protocol (i.e., the KOY protocol [22]), where two parties, who share a password, exchange messages to establish a common secret key. Their basic two-server protocol is secure against a passive (i.e., "honest-but-curious") adversary who has access to one of the servers throughout the protocol execution, but cannot cause this server to deviate from its prescribed behavior. In [23], Katz et al. also showed how to modify their basic protocol so as to achieve security against an active adversary who may cause a corrupted server to deviate arbitrarily from the protocol. The core of their protocol is the KOY protocol. The client looks like running two KOY protocols with two servers in parallel. However, each server must perform a total of roughly 80 exponentiations (i.e., each server's work is increased by a factor of roughly 6 as compared to the basic protocol [23]).

In [35], a security model for ID2S PAKE protocol was given and a compiler that transforms any two-party PAKE protocol to an ID2S PAKE protocol was proposed on the basis of the Cramer-Shoup public key encryption scheme [13] and any identity-based encryption scheme, such as the Waters' scheme [28].

Our Contribution. In this paper, we propose a new compiler for ID2S PAKE protocol based on any identity-based signature scheme (IBS), such as the Paterson et al.'s scheme [25]. The basic idea is: The client splits its password into two shares and each server keeps one share of the password in addition to a private key related to its identity for signing. In key exchange, each server sends the client its public key for encryption with its identity-based signature on it. The signature can be verified by the client on the basis of the identity of the server. If the signature is genuine, the client submits to the server one share of the password encrypted with the public key of the server. With the decryption keys, both servers can derive the same one-time password, by which the two servers can run a two-party PAKE protocol to authenticate the client.

In addition, we generalize the compiler based on IBE in [35] by replacing the Cramer-Shoup public key encryption scheme with any public key encryption scheme. Unlike the compiler based on IBS, the compiler based on IBE assumes that each server has a private key related to its identity for decryption. In key exchange, the client sends to each server one share of the password encrypted according to the identity of the server. In addition, a one-time public key encryption scheme is used to protect the messages (containing the password information) from the servers to the client. The one-time public key is generated by the client and sent to the servers along with the password information in the first phase.

In the identity-based cryptography, the decryption key or the signing key of a server is usually generated by a Private Key Generator (PKG). Therefore the PKG can decrypt any messages encrypted with the identity of the server or sign any document on behalf of the server. As mentioned in [7], using standard techniques from threshold cryptography, the PKG can be distributed so that the master-key is never available in a single location. Like [35], our strategy is to employ multiple PKGs which cooperate to generate the decryption key or the signing key for the server. As long as one of the PKGs is honest to follow the protocol, the decryption key or the signing key for the server is known only to the server. Since we can assume that the two servers in two-server PAKE never collude, we can also assume that at least one of the PKGs do not collude with other PKGs.

Based on this assumption, we provide a rigorous proof of security for our compilers. The two compilers do not rely on the random oracle model as long as the underlying primitives themselves do not rely on it. For example, by using the KOY protocol [22] and the Paterson et al.'s IBS scheme [25] and the Cramer-Shoup public key encryption scheme [13], the compiler based on IBS can construct an ID2S PAKE protocol with provable security in the standard model. By using the KOY protocol [22] and the Waters IBE scheme [28] and the Cramer-Shoup public key encryption scheme [13], the compiler based on IBE can construct an

ID2S PAKE protocol with provable security in the standard model.

We also compare our ID2S PAKE protocols with the Katz et al.'s two-server PAKE protocol [23] with provable security in the standard model. The Katz et al.'s protocol is password-only, where the client needs to remember the password only and refer to common public parameters, and each server, having a public and private key pair, and keeps a share of the password. Our protocols are identity-based, where the client needs to remember the password in addition to the meaningful identities of the two servers, and refer to common public parameters, including the master public key, and each server, having a private key related to his identity, keeps a share of the password.

In terms of the setting and the client performance, the Katz et al.'s protocol is superior to our protocols. However, in the Katz et al.'s protocol, each server performs approximately six times the amount of the work as the KOY protocol, whereas in our protocols, each server performs the same amount of work as the KOY protocol in addition to one identity-based decryption (or signature) and one public key encryption (or decryption).

We have implemented our ID2S PAKE protocols. Our experiments show that our protocols save from 22% to 66% of computation in each server, compared with the Katz et al.'s protocol. The server performance is critical to the performance of the whole protocol when the servers provide services to a great number of clients concurrently. In addition, our experiments show that less than one second is needed for the client to execute our protocols.

Organization. In Section 2, we describe the security model for ID2S PAKE protocol given in [35]. In Section 3, we present our new ID2S PAKE compilers. After that, in Section 4, a rigorous proof of security for our protocols is provided. In Section 5, we analyze the performance of our protocols and compare them with the Katz's protocol by experiments. We conclude this paper in Section 6.

2 Definitions

A formal model of security for two-server PAKE was given by Katz et al. [23] (based on the MacKenzie et al.'s model for PKI-based PAKE [24]). Boneh and Franklin [7] defined chosen ciphertext security for IBE under chosen identity attack. Combining the two models, a model for ID2S PAKE protocol was given in [35] and can be described as follows.

Participants, Initialization and Passwords. An ID2S PAKE protocol involves three kinds of protocol participants: (1) A set of clients (denoted as Client), each of which requests services from servers on the network; (2) A set of servers (denoted as Server), each of which provides services to clients on the network; (3) A group of Private Key Generators (PKGs), which generate public parameters and corresponding private keys for servers.

We assume that $\mathsf{ClientServerTriple}$ is the set of triples of the client and two servers, where the client is authorized to use services provided by the two servers, $\mathsf{Client} \bigcap \mathsf{Server} = \emptyset$, $\mathsf{User} = \mathsf{Client} \bigcup \mathsf{Server}$, any $\mathsf{PKG} \notin \mathsf{User}$, and $\mathsf{ClientServerTriple} \subseteq \mathsf{Client} \times \mathsf{Server} \times \mathsf{Server}$.

Prior to any execution of the protocol, we assume that an initialization phase occurs. During initialization, the PKGs cooperate to generate public parameters for the protocol, which are available to all participants, and private keys for servers, which are given to the appropriate servers. The user may keep the public parameter in a personal device, such as a smart card or a USB flash drive. When the PKGs generate the private key for a server, each PKG generates and sends a private key component to the server via a secure channel. The server then derives its private key by combining all private key components from all PKGs. **We assume that at least one of PKGs is honest to follow the protocol**. Therefore, the private key of the server is known to the server only.

For any triple $(C, A, B) \in \mathsf{ClientServerTriple}$, we assume that the client C chooses its password pw_C independently and uniformly at random from a "dictionary" $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ of size N, where $\mathcal{D} \subset Z_q$, N is a fixed constant which is independent of any security parameter, and q is a large prime. The password is then split into two shares $\mathsf{pw}_{C,A}$ and $\mathsf{pw}_{C,B}$ and stored at the two servers A and B, respectively, for authentication. **We assume that the two servers never collude to determine the password of the client**. The client C needs to remember pw_C to log into the servers A and B.

For simplicity, we assume that each client C shares its password pw_C with exactly two servers A and B. In this case, we say that servers A and B are associated with C. A server may be associated with multiple clients.

Execution of the Protocol. In the real world, a protocol determines how users behave in response to input from their environments. In the formal model, these inputs are provided by the adversary. Each user is assumed to be able to execute the protocol multiple times (possibly concurrently) with different partners. This is modeled by allowing each user to have unlimited number of instances (please refer to [3]) with which to execute the protocol. We denote instance i of user U as U^i. A given instance may be used only once. The adversary is given oracle access to these different instances. Furthermore, each instance maintains (local) state which is updated during the course of the experiment. In particular, each instance U^i is associated with the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase.

— $\mathsf{sid}_U^i, \mathsf{pid}_U^i$ and sk_U^i are variables containing the session identity, partner identity, and session key for an instance U^i, respectively. Computation of the session key is, of course, the ultimate goal of the protocol. The session identity is simply a way to keep track of the different executions of a particular user U. Without loss of generality, we simply let this be the (ordered) concatenation of all messages sent and received by instance U^i. The partner identity denotes the identity of the user with whom U^i believes it is interacting. For a client C, sk_C^i consists of a pair $(\mathsf{sk}_{C,A}^i, \mathsf{sk}_{C,B}^i)$, which are the two keys shared with servers A and B, respectively.
— acc_U^i and term_U^i are boolean variables denoting whether a given instance U^i has been accepted or terminated, respectively. Termination means that the given instance has done receiving and sending messages, acceptance indicates successful termination. In our case, acceptance means that the instance is sure that it has established a session key with its intended partner; thus, when an instance U^i has been

accepted, sid_U^i, pid_U^i and sk_U^i are no longer NULL.
— state_U^i records any state necessary for execution of the protocol by U^i.
— used_U^i is a boolean variable denoting whether an instance U^i has begun executing the protocol. This is a formalism which will ensure each instance is used only once.

The adversary $\mathcal{A}$ is assumed to have complete control over all communications in the network (between the clients and servers, and between servers and servers) and the adversary's interaction with the users (more specifically, with various instances) is modelled via access to oracles. The state of an instance may be updated during an oracle call, and the oracle's output may depend upon the relevant instance. The oracle types include:

— $\mathsf{Send}(C, i, A, B, M)$ – This sends message M to a client instance C^i, supposedly from two servers A and B. Assuming $\mathsf{term}_C^i = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of C^i (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_C^i, \mathsf{pid}_C^i, \mathsf{acc}_C^i$, and term_C^i. This oracle call models the active attack to a protocol. If M is empty, this query represents a prompt for C to initiate the protocol.
— $\mathsf{Send}(S, i, U, M)$ – This sends message M to a server instance S^i, supposedly from a user U (either a client or a server). Assuming $\mathsf{term}_S^i = \mathsf{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of S^i (i.e., the message sent by the instance) is given to the adversary, who receives the updated values of $\mathsf{sid}_S^i, \mathsf{pid}_S^i, \mathsf{acc}_S^i$, and term_S^i. If S is corrupted, the adversary also receives the entire internal state of S. This oracle call also models the active attack to a protocol.
— $\mathsf{Execute}(C, i, A, j, B, k)$ – If the client instance C^i and the server instances A^j and B^k have not yet been used (where $(C, A, B) \in \mathsf{ClientServerTriple}$), this oracle executes the protocol between these instances and outputs the transcript of this execution. This oracle call represents passive eavesdropping of a protocol execution. In addition to the transcript, the adversary receives the values of sid, pid, acc, and term for client and server instances, at each step of protocol execution. In addition, if $S \in \{A, B\}$ is corrupted, the adversary is given the entire internal state of S.
— $\mathsf{Corrupt}(S)$ – This sends the private key of the server S in addition to all password information stored in the server S to the adversary. This oracle models possible compromising of a server due to, for example, hacking into the server.
— $\mathsf{Corrupt}(C)$ – This query allows the adversary to learn the password of the client C, which models the possibility of subverting a client by, for example, witnessing a user typing in his password, or installing a "Trojan horse" on his machine.
— $\mathsf{Reveal}(U, U', i)$ – This outputs the current value of session key $\mathsf{sk}_{U,U'}^i$ held by instance U^i if $\mathsf{acc}_U^i = \mathsf{TRUE}$, where $U' \in \mathsf{pid}_U^i$. This oracle call models possible leakages of session keys due to, for example, improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.
— $\mathsf{Test}(U, U', i)$ – This oracle does not model any real-world capability of the adversary, but is instead used to define security. Assume $U' \in \mathsf{pid}_U^i$, if $\mathsf{acc}_U^i = \mathsf{TRUE}$, a random bit b is generated. If $b = 0$, the adversary is given $\mathsf{sk}_{U,U'}^i$, and if $b = 1$ the adversary is given a random session key. The adversary is allowed only a single Test query, at any time during its execution.

Partnering. Let $(C, A, B) \in \mathsf{ClientServerTriple}$. For the client instance C^i, let $\mathsf{sid}_C^i = (\mathsf{sid}_{C,A}^i, \mathsf{sid}_{C,B}^i)$, where $\mathsf{sid}_{C,A}^i$ (resp., $\mathsf{sid}_{C,B}^i$) denotes the ordered sequence of messages sent to / from the client C and the server A (resp., server B). For the server instance A^j, let $\mathsf{sid}_A^j = (\mathsf{sid}_{A,C}^j, \mathsf{sid}_{A,B}^j)$, where $\mathsf{sid}_{A,C}^j$ denotes the ordered sequence of messages sent to / from the server A and the client C, and $\mathsf{sid}_{A,B}^j$ denotes the ordered sequence of message sent to / from the server A and the server B. We say that instances C^i and A^j are partnered if (1) $\mathsf{sid}_{C,A}^i = \mathsf{sid}_{A,C}^j \neq \mathsf{NULL}$ and (2) $A \in \mathsf{pid}_C^i$ and $C \in \mathsf{pid}_A^j$. We say that instances A^j and B^k are partnered if (1) $\mathsf{sid}_{A,B}^j = \mathsf{sid}_{B,A}^k \neq \mathsf{NULL}$ and (2) $A \in \mathsf{pid}_B^k$ and $B \in \mathsf{pid}_A^j$.

Correctness. To be viable, a key exchange protocol must satisfy the following notion of correctness: If a client instance C^i and server instances A^j and B^k run an honest execution of the protocol with no interference from the adversary, then $\mathsf{acc}_C^i = \mathsf{acc}_A^j = \mathsf{acc}_B^k = \mathsf{TRUE}$, and $\mathsf{sk}_{C,A}^i = \mathsf{sk}_{A,C}^j$, $\mathsf{sk}_{C,B}^i = \mathsf{sk}_{B,C}^k$ and $\mathsf{sk}_{C,A}^i \neq \mathsf{sk}_{C,B}^i$. Note that a correct protocol may not be secure. The security of a protocol is defined as follows.

Freshness. To formally define the adversary's success we need to define a notion of freshness for a session key, where freshness of a key is meant to indicate that the adversary does not trivially know the value of the session key. We say a session key $\mathsf{sk}_{U,U'}^i$ is fresh if (1) both U and U' are not corrupted; (2) the adversary never queried $\mathsf{Reveal}(U, U', i)$; (3) the adversary never queried $\mathsf{Reveal}(U', U, j)$ where U^i and U'^j are partnered.

Advantage of the Adversary. Informally, the adversary succeeds if it can guess the bit b used by the Test oracle. We say an adversary $\mathcal{A}$ succeeds if it makes a single query $\mathsf{Test}(U, U', i)$ to a fresh instance U^i, with $\mathsf{acc}_U^i = \mathsf{TRUE}$ at the time of this query, and outputs a single bit b' with $b' = b$ (recall that b is the bit chosen by the Test oracle). We denote this event by Succ. The advantage of adversary $\mathcal{A}$ in attacking protocol P is then given by $\mathsf{Adv}_{\mathcal{A}}^P(k) = 2 \cdot \Pr[\mathsf{Succ}] - 1$, where the probability is taken over the random coins used by the adversary and the random coins used during the course of the experiment (including the initialization phase).

An adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack. A protocol is secure if this is the best an adversary can do. The on-line attacks correspond to Send queries. Formally, each instance for which the adversary has made a Send query counts as one on-line attack. Instances with which the adversary interacts via Execute are not counted as on-line attacks. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion.

Definition 1. Protocol P is a secure ID2S PAKE protocol if, for all dictionary size N and for all PPT adversaries $\mathcal{A}$ making at most $Q(k)$ on-line attacks, there exists a negligible function $\varepsilon(\cdot)$ such that $\mathsf{Adv}_{\mathcal{A}}^P(k) \leq Q(k)/N + \varepsilon(k)$.

3 ID2S PAKE PROTOCOLS

In this section, we present two compilers transforming any two-party PAKE protocol P to an ID2S PAKE protocol P' with identity-based cryptography. The first compiler is built on identity-based signature (IBS) and the second compiler is based on identity-based encryption (IBE).

3.1 ID2S PAKE Based on IBS

3.1.1 Protocol Description

We need an identity-based signature scheme (IBS) as our cryptographic building block. A high-level description of our compiler is given in Fig. 1, in which the client C and two servers A and B establish two authenticated keys, respectively. If we remove authentication elements from our compiler, our key exchange protocol is essentially the Diffie-Hellman key exchange protocol [14]. We present the protocol by describing initialization and execution.

Initialization. Given a security parameter $k \in \mathbb{N}$ (the set of all natural number), the initialization includes:

Parameter Generation: On input k, (1) m PKGs cooperate to run $\mathsf{Setup}^{\mathsf{P}}$ of the two-party PAKE protocol P to generate system parameters, denoted as $\mathsf{params}^{\mathsf{P}}$. (2) m PKGs cooperate to run $\mathsf{Setup}^{\mathsf{IBS}}$ of the IBS scheme to generate public system parameters for the IBS scheme, denoted as $\mathsf{params}^{\mathsf{IBS}}$ (including a subgroup $\mathbb{G}$ of the additive group of points of an elliptic curve), and the secret $\mathsf{master\text{-}key}^{\mathsf{IBS}}$. (3) m PKGs choose a public key encryption scheme E, e.g., [13], whose plaintext group is a large cyclic group G with a prime order q and a generator g and select two hash functions, $H_1 : \{0,1\}^* \rightarrow Z_n^*$, where n is the order of $\mathbb{G}$, and $H_2 : \{0,1\}^* \rightarrow Z_q^*$, from a collision-resistant hash family. The public system parameters for the protocol P' is $\mathsf{params} = \mathsf{params}^{\mathsf{P},\mathsf{IBS},\mathsf{E}} \bigcup \{(G, q, g), (H_1, H_2)\}$ and the secret $\mathsf{master\text{-}key}^{\mathsf{IBS}}$ is secretly shared by the PKGs in a manner that any coalition of PKGs cannot determine $\mathsf{master\text{-}key}^{\mathsf{IBS}}$ as long as one of the PKGs is honest to follow the protocol.

Remark. Taking the Paterson-Schuldt IBS scheme [25] for example, m PKGs agree on randomly chosen $\mathcal{G}, \mathcal{G}_2 \in \mathbb{G}$ and each PKG randomly chooses $\alpha_i \in Z_p$ and broadcast $\mathcal{G}^{\alpha_i}$ with a zero-knowledge proof of knowing α_i and a signature. Then we can set $\mathcal{G}_1 = \mathcal{G}^{\sum_i \alpha_i}$ as the public master key and the secret $\mathsf{master\text{-}key}^{\mathsf{IBS}} = \mathcal{G}_2^{\sum_i \alpha_i}$. The secret master key is privately shared among m PKGs and unknown to anyone even if $m-1$ PKGs maliciously collude.

Key Generation: On input the identity S of a server $S \in \mathsf{Server}$, $\mathsf{params}^{\mathsf{IBS}}$, and the secret sharing $\mathsf{master\text{-}key}^{\mathsf{IBS}}$, PKGs cooperate to run $\mathsf{Extract}^{\mathsf{IBS}}$ of the IBS scheme and generate a private (signing) key for S, denoted as d_S, in a manner that any coalition of PKGs cannot determine d_S as long as one of the PKGs is honest to follow the protocol.

Remark. In the Paterson-Schuldt IBS scheme with m PKGs , each PKG computes one component of the private key for a server S, i.e., $(\mathcal{G}_2^{\alpha_i} H(S)^{r_i}, \mathcal{G}^{r_i})$, where H is the Waters' hash function, and sends it to the server via a secure channel. Combining all components, the server can construct its private key $d_S = (\mathcal{G}_2^{\sum_i \alpha_i} H(S)^{\sum_i r_i}, \mathcal{G}^{\sum_i r_i})$, which is known to the server only even if $m-1$ PKGs maliciously collude. In addition, the identity of a server is public, meaningful, like an e-mail address, and easy to remember or keep. Anyone can write down the identity of a server on a note.

Password Generation: On input a triple $(C, A, B) \in \mathsf{Client}\,\mathsf{Server}\mathsf{Triple}$, a string pw_C, the password, is uniformly drawn from the dictionary $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ by the client C, and randomly split into $\mathsf{pw}_{C,A}$ and $\mathsf{pw}_{C,B}$ such that $\mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} = \mathsf{pw}_C (mod\ q)$. $g^{\mathsf{pw}_{C,A}}$ and $g^{\mathsf{pw}_{C,B}}$ are stored in the servers A and B, respectively. We implicitly assume that $N < min(n, q)$, which will certainly be true in practice.

Protocol Execution. Given a triple $(C, A, B) \in \mathsf{Client}\,\mathsf{Server}\mathsf{Triple}$, the client C (knowing its password pw_C) runs the protocol P' with the two servers A (knowing $g^{\mathsf{pw}_{C,A}}$ and its private key d_A) and B (knowing $g^{\mathsf{pw}_{C,B}}$ and its private key d_B) to establish two session keys, respectively, as shown in Fig. 1.

At first, the client C randomly chooses an integer r_c from Z_q^* and computes $W_c = g^{r_c}$ and broadcasts $\mathsf{msg} = \langle C, W_c \rangle$ to the servers A and B.

Remark. To facilitate the communications between the client and two servers, a gateway may be used to forward messages between the client and the two servers. In this case, the client needs to communicate with the gateway only.

After receiving msg from the client, the server X (either A or B) randomly generates a public and private key pair (pk_x, sk_x) (where x is either a or b) for the public key encryption scheme E, and randomly chooses an integer r_x from Z_q^* and computes

$$W_x = g^{r_x}, h_x = H_1(X, W_x, C, W_x, pk_x), S_x = \mathsf{Sign}(h_x, d_X)$$

where S_x denotes an identity-based signature of X on h_x. Then the server X replies to the client C with $\mathsf{msg}_X = \langle X, W_x, pk_x, S_x \rangle$.

Remark. The public and private key pair for the public key encryption scheme E can be generated once only and used repeatedly.

When getting the responses $\mathsf{msg}_A, \mathsf{msg}_B$ from the servers A and B, the client C computes

$$h'_a = H_1(A, W_a, C, W_c, pk_a), h'_b = H_1(B, W_b, C, W_c, pk_b)$$

and verifies the two signatures S_a, S_b on the basis of the identities of the servers A and B. If

$$(\mathsf{Verify}(h'_a, S_a, A) = \mathsf{TRUE}) \wedge (\mathsf{Verify}(h'_b, S_b, B) = \mathsf{TRUE})$$

the client C sets $\mathsf{acc}_C = \mathsf{TRUE}$ and computes two session keys $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$.

Furthermore, the client C randomly chooses pw_1 from Z_q^* and computes

$$\mathsf{pw}_2 = \mathsf{pw}_C - \mathsf{pw}_1 (mod\ q)$$

$$h_1 = H_2(C, W_c, A, W_a), h_2 = H_2(C, W_c, B, W_b).$$

Next, according to the public keys pk_a, pk_b from the servers A and B, the client C performs the public key encryptions

$$E_a = \mathsf{E}(g^{\mathsf{pw}_1 h_1^{-1}}, pk_a), E_b = \mathsf{E}(g^{\mathsf{pw}_2 h_2^{-1}}, pk_b)$$

where E denotes the encryption algorithm. Then, the client sends $\mathsf{msg}_1 = \langle C, E_a \rangle$ and $\mathsf{msg}_2 = \langle C, E_b \rangle$ to the two servers A and B, respectively.

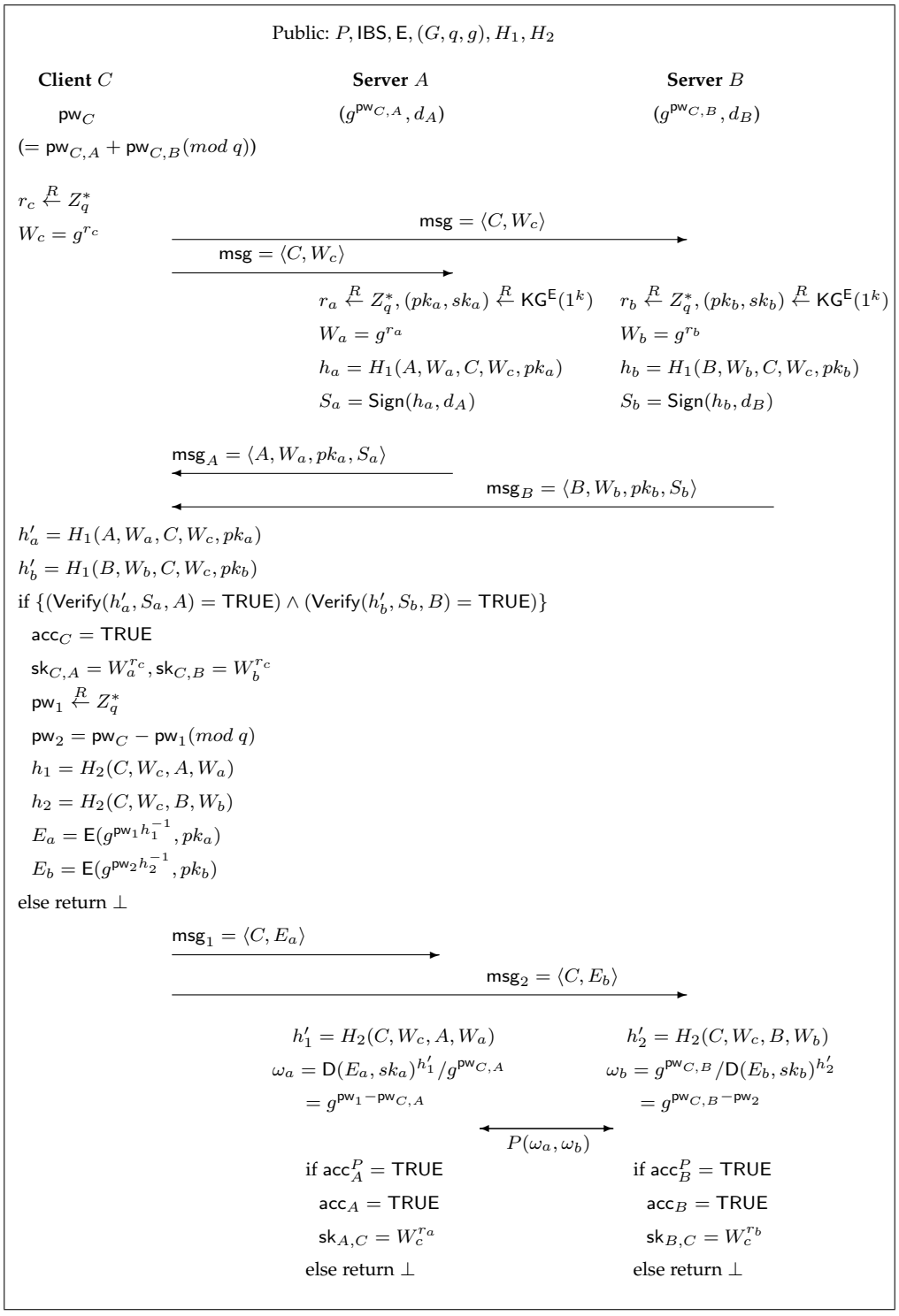

Fig. 1. ID2S PAKE protocol P' based on IBS

After receiving msg_1 from C, the server A computes

$$h_1' = H_2(C, W_c, A, W_a), \omega_a = \mathsf{D}(E_a, sk_a)^{h_1'} / g^{\mathsf{pw}_{C,A}},$$

where D denotes the decryption algorithm.

After receiving msg_2 from C, the server B computes

$$h_2' = H_2(C, W_c, B, W_b), \omega_b = g^{\mathsf{pw}_{C,B}} / \mathsf{D}(E_b, sk_b)^{h_2'}.$$

Because $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} (mod\ q)$ and $\mathsf{pw}_C = \mathsf{pw}_1 + \mathsf{pw}_2 (mod\ q)$, we have $\omega_a = g^{\mathsf{pw}_1 - \mathsf{pw}_{C,A}} = g^{\mathsf{pw}_{C,B} - \mathsf{pw}_2} = \omega_b$.

Using ω_a and ω_b as one-time password, the servers A and B run a two-party PAKE protocol P to establish a session key. If the server A accepts the session key as an authenticated key according to P (i.e., $\mathsf{acc}_A^P = \mathsf{TRUE}$), the server A sets $\mathsf{acc}_A = \mathsf{TRUE}$ and computes the session key $\mathsf{sk}_{A,C} = W_c^{r_a}$. If the server B accepts the session key as an authenticated key according to P (i.e., $\mathsf{acc}_B^P = \mathsf{TRUE}$), the server B sets $\mathsf{acc}_B = \mathsf{TRUE}$ and computes the session key $\mathsf{sk}_{B,C} = W_c^{r_b}$.

3.1.2 Correctness

Assume that a client instance C^i and server instances A^j and B^k run an honest execution of the protocol P' with no interference from the adversary and the two-party PAKE P has the correctness property.

With reference to Fig. 1, we have $h_a' = h_a, h_b' = h_b$ and the signatures are genuine. Therefore, the client C computes two session keys, i.e., $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$, and lets $\mathsf{acc}_C^i = \mathsf{TRUE}$.

With reference to Fig. 1, the server instances A^j and B^k are able to derive the same one-time password ω_a ($= \omega_b$). Because P has the correctness property, after running P based on ω_a and ω_b, the server instances A^j and B^k accept the established session key as an authenticated key. This indicates that the client C has provided a correct password pw_C. Next, the server instances A^j and B^k compute the session keys with the client C, i.e., $\mathsf{sk}_{A,C} = W_c^{r_a}$ and $\mathsf{sk}_{B,C} = W_c^{r_b}$, and let $\mathsf{acc}_A^j = \mathsf{TRUE}$ and $\mathsf{acc}_B^k = \mathsf{TRUE}$.

Since $W_c = g^{r_c}, W_a = g^{r_a}, W_b = g^{r_b}$, we have $\mathsf{sk}_{C,A} = W_a^{r_c} = g^{r_a r_c} = W_c^{r_a} = \mathsf{sk}_{A,C}$ and $\mathsf{sk}_{C,B} = W_b^{r_c} = g^{r_b r_c} = W_c^{r_b} = \mathsf{sk}_{B,C}$. In addition, because r_a, r_b are chosen randomly, the probability of $\mathsf{sk}_{C,A} = \mathsf{sk}_{C,B}$ is negligible. Therefore, our protocol has the correctness property.

3.1.3 Explicit Authentication

By verifying the signatures S_a, S_b with the identities of the servers A and B, the client C can make sure that its intended servers A and B are authentic and the computed session keys $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$ are authentic.

By running the two-party PAKE protocol P based on w_a (derived by $\mathsf{pw}_{C,A}$) and w_b (derived by $\mathsf{pw}_{C,B}$), the two servers A and B can verify if the client C provides a password pw_C such that $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} (mod\ q)$. This shows that when $\mathsf{acc}_A^j = \mathsf{TRUE}$, the server A knows that its intended client C and server B are authentic. Our protocol achieves the implicit authentication. Using the hash function like [6], [35], however, it is easy to add explicit authentication to any protocol achieving implicit authentication.

Remark. After the execution of our protocol, the password in the client cache must be deleted like SSL-based Internet banking.

3.2 ID2S PAKE Based on IBE

3.2.1 Protocol Description

A high-level description of our compiler based on identity-based encryption (IBE) is given in Fig. 2. We present the protocol by describing initialization and execution.

Initialization. Given a security parameter $k \in \mathbb{N}$, the initialization includes:

Parameter Generation: On input k, (1) m PKGs cooperate to run $\mathsf{Setup}^{\mathsf{P}}$ of the two-party PAKE protocol P to generate system parameters, denoted as $\mathsf{params}^{\mathsf{P}}$. (2) m PKGs cooperate to run $\mathsf{Setup}^{\mathsf{IBE}}$ of the IBE scheme to generate public system parameters for the IBE scheme, denoted as $\mathsf{params}^{\mathsf{IBE}}$, and the secret $\mathsf{master\text{-}key}^{\mathsf{IBE}}$. Assume that $\mathcal{G}$ is a generator of IBE plaintext group $\mathbb{G}$ with an order n. (3) m PKGs choose a public key encryption scheme E, e.g., [13], whose plaintext group is a large cyclic group G with a prime order q and a generator g and select two hash functions, $H_1 : \{0,1\}^* \to Z_n^*$ and $H_2 : \{0,1\}^* \to Z_q^*$, from a collision-resistant hash family. The public system parameters for the protocol P' is $\mathsf{params} = \mathsf{params}^{\mathsf{P,IBE,E}} \bigcup \{(\mathbb{G}, \mathcal{G}, n), (G, q, g), (H_1, H_2)\}$ and the secret $\mathsf{master\text{-}key}^{\mathsf{IBE}}$ is secretly shared by the PKGs in a manner that any coalition of PKGs cannot determine $\mathsf{master\text{-}key}^{\mathsf{IBE}}$ as long as one of the PKGs is honest to follow the protocol.

Key Generation: On input the identity S of a server $S \in \mathsf{Server}$, $\mathsf{params}^{\mathsf{IBE}}$, and the secret sharing $\mathsf{master\text{-}key}^{\mathsf{IBE}}$, PKGs cooperate to run $\mathsf{Extract}^{\mathsf{IBE}}$ of the IBE scheme and generate a private (decryption) key for S, denoted as d_S, in a manner that any coalition of PKGs cannot determine d_S as long as one of the PKGs is honest to follow the protocol.

Password Generation: On input a triple $(C, A, B) \in \mathsf{Client\ ServerTriple}$, a string pw_C, the password, is uniformly drawn from the dictionary $\mathcal{D} = \{\mathsf{pw}_1, \mathsf{pw}_2, \cdots, \mathsf{pw}_N\}$ by the client C, and randomly split into $\mathsf{pw}_{C,A}$ and $\mathsf{pw}_{C,B}$ such that $\mathsf{pw}_{C,A} + \mathsf{pw}_{C,B} = \mathsf{pw}_C (mod\ n)$, and $\mathsf{pw}_{C,A}^*$ and $\mathsf{pw}_{C,B}^*$ such that $\mathsf{pw}_{C,A}^* + \mathsf{pw}_{C,B}^* = \mathsf{pw}_C (mod\ q)$. ($\mathcal{G}^{\mathsf{pw}_{C,A}}$, $g^{\mathsf{pw}_{C,A}^*}$) and ($\mathcal{G}^{\mathsf{pw}_{C,B}}$, $g^{\mathsf{pw}_{C,B}^*}$) are stored in severs A and B, respectively. We implicitly assume that $N < min(n, q)$, which will certainly be true in practice.

Protocol Execution. Given a triple $(C, A, B) \in \mathsf{Client\ ServerTriple}$, the client C (knowing its password pw_C) runs the protocol P' with the two servers A (knowing $\mathcal{G}^{\mathsf{pw}_{C,A}}$, $g^{\mathsf{pw}_{C,A}^*}$ and its private key d_A) and B (knowing $\mathcal{G}^{\mathsf{pw}_{C,B}}$, $g^{\mathsf{pw}_{C,B}^*}$ and its private key d_B) to establish two session keys, respectively, as shown in Fig. 2.

At first, the client randomly chooses pw_1 from Z_n^* and computes $\mathsf{pw}_2 = \mathsf{pw}_C - \mathsf{pw}_1 (mod\ n)$. Next the client C randomly generates a one-time public and private key pair (pk, sk) for the public key encryption scheme E, and randomly chooses an integer r_c from Z_q^* and computes

$$W_c = g^{r_c}, h = H_1(C, W_c, pk).$$

Next, according to the identities of the two servers A and B, the client C performs the identity-based encryptions

$$E_a = \mathsf{IBE}(\mathcal{G}^{\mathsf{pw}_1 h^{-1}}, A), E_b = \mathsf{IBE}(\mathcal{G}^{\mathsf{pw}_2 h^{-1}}, B).$$

Then, the client sends $\mathsf{msg}_1 = \langle C, W_c, pk, E_a \rangle$ and $\mathsf{msg}_2 = \langle C, W_c, pk, E_b \rangle$ to the two servers A and B, respectively.

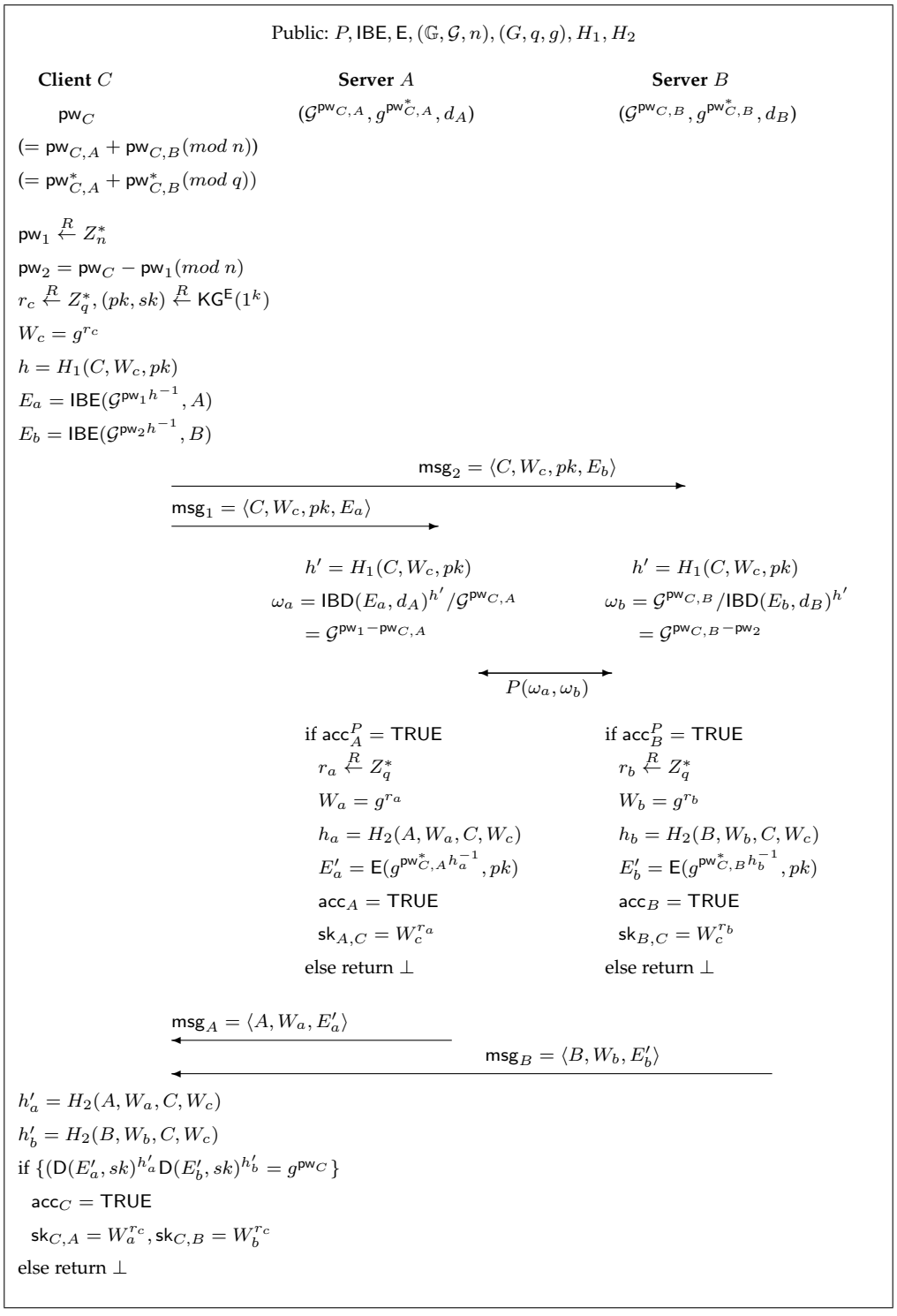

Fig. 2. ID2S PAKE protocol P' based on IBE

After receiving msg_1 from C, the server A computes

$$h' = H_1(C, W_c, pk), \omega_a = \mathsf{IBD}(E_a, d_A)^{h'} / \mathcal{G}^{\mathsf{pw}_{C,A}},$$

where IBD denotes identity-based decryption.

After receiving msg_2 from C, the server B computes

$$h' = H_1(C, W_c, pk), \omega_b = \mathcal{G}^{\mathsf{pw}_{C,B}} / \mathsf{IBD}(E_a, d_B)^{h'},$$

Because $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B}(mod\ n)$ and $\mathsf{pw}_C = \mathsf{pw}_1 + \mathsf{pw}_2(mod\ n)$, we have $\omega_a = \mathcal{G}^{\mathsf{pw}_1 - \mathsf{pw}_{C,A}} = \mathcal{G}^{\mathsf{pw}_{C,B} - \mathsf{pw}_2} = \omega_b$.

Using ω_a and ω_b as one-time password, the servers A and B run a two-party PAKE protocol P to establish a session key. If the server X (either A or B) accepts the session key as an authenticated key according to P (i.e., $\mathsf{acc}_X^P = \mathsf{TRUE}$), it randomly chooses an integer r_x (where x is either a or b) from Z_q^* and computes

$$W_x = g^{r_x}, h_x = H_2(X, W_x, C, W_c),$$

$$E'_x = \mathsf{E}(g^{\mathsf{pw}^*_{C,X} h_x^{-1}}, pk), \mathsf{sk}_{X,C} = W_c^{r_x}$$

where $\mathsf{sk}_{X,C}$ is the session key between X and C and E'_x is the encryption of $g^{\mathsf{pw}^*_{C,X} h_x^{-1}}$. Then the server X sets $\mathsf{acc}_X = \mathsf{TRUE}$ and replies to the client C with $\mathsf{msg}_X = \langle X, W_x, E'_x \rangle$.

Finally, after receiving msg_A and msg_B, C computes

$$h'_a = H_2(A, W_a, C, W_c), h'_b = H_2(B, W_b, C, W_c),$$

and check if $\mathsf{D}(E_1, sk)^{h'_a} \mathsf{D}(E_2, sk)^{h'_b} = g^{\mathsf{pw}_C}$. If so, the client C sets $\mathsf{acc}_C = \mathsf{TRUE}$ and computes two session keys $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$.

Remark. Our IBE-based protocol needs less rounds of communication than our IBS-based protocol. This can be seen by comparing Fig. 1 and Fig. 2.

3.2.2 Correctness

Assume that a client instance C^i and server instances A^j and B^k run an honest execution of the protocol P' with no interference from the adversary and the two-party PAKE P has the correctness property.

With reference to Fig. 2, the server instances A^j and B^k are able to derive the same one-time password ω_a $(= \omega_b)$. Because P has the correctness property, after running P based on ω_a and ω_b, the server instances A^j and B^k accept the established session key as an authenticated key. This indicates that the client C has provided a correct password pw_C. Next, the server instances A^j and B^k compute the session keys with the client C, i.e., $\mathsf{sk}_{A,C} = W_c^{r_a}$ and $\mathsf{sk}_{B,C} = W_c^{r_b}$, and let $\mathsf{acc}_A^j = \mathsf{TRUE}$ and $\mathsf{acc}_B^k = \mathsf{TRUE}$.

With reference to Fig. 2, we have $h'_a = h_a, h'_b = h_b$, and

$$\begin{aligned} \mathsf{D}(E_1, sk)^{h'_1} \mathsf{D}(E_2, sk)^{h'_b} &= (g^{\mathsf{pw}^*_{C,A} h_a^{-1}})^{h'_a} (g^{\mathsf{pw}^*_{C,B} h_b^{-1}})^{h'_b} \\ &= g^{\mathsf{pw}^*_{C,A} + \mathsf{pw}^*_{C,B}} = g^{\mathsf{pw}_C} \end{aligned}$$

Therefore, the client instance C^i computes two session keys, i.e., $\mathsf{sk}_{C,A} = W_a^{r_c}, \mathsf{sk}_{C,B} = W_b^{r_c}$, and lets $\mathsf{acc}_C^i = \mathsf{TRUE}$. Since $W_c = g^{r_c}, W_a = g^{r_a}, W_b = g^{r_b}$, we have $\mathsf{sk}_{C,A} = W_a^{r_c} = g^{r_a r_c} = W_c^{r_a} = \mathsf{sk}_{A,C}$ and $\mathsf{sk}_{C,B} = W_b^{r_c} = g^{r_b r_c} = W_c^{r_b} = \mathsf{sk}_{B,C}$. In addition, because r_a, r_b are chosen randomly, the probability of $\mathsf{sk}_{C,A} = \mathsf{sk}_{C,B}$ is negligible. Therefore, our protocol has the correctness property.

3.2.3 Explicit Authentication

By running the two-party PAKE protocol P based on w_a (derived by $\mathsf{pw}_{C,A}$) and w_b (derived by $\mathsf{pw}_{C,B}$), the two servers A and B can verify if the client C provides a password pw_C such that $\mathsf{pw}_C = \mathsf{pw}_{C,A} + \mathsf{pw}_{C,B}(mod\ n)$. In addition, by checking that $\mathsf{D}(E_1, sk)^{h'_a} \mathsf{D}(E_2, sk)^{h'_b} = g^{\mathsf{pw}_C}$ (involving pw_C), the client C can verify if the two servers provide two shares of the password, $\mathsf{pw}^*_{C,A}$ and $\mathsf{pw}^*_{C,B}$, such that $\mathsf{pw}_C = \mathsf{pw}^*_{C,A} + \mathsf{pw}^*_{C,B}(mod\ q)$. This shows that when $\mathsf{acc}_A^j = \mathsf{TRUE}$, the server A knows that its intended client C and server B are authentic, and when $\mathsf{acc}_C^i = \mathsf{TRUE}$, the client C knows that its intended servers A and B are authentic. Our protocol achieves the implicit authentication. Using the hash function like [6], [35], however, it is easy to add explicit authentication to any protocol achieving implicit authentication.

4 Proof of Security

Based on the security model defined in Section 2, we provide a rigorous proof of security for our compilers in this section.

4.1 Security of ID2S PAKE Protocol Based on IBS

Theorem 1. Assuming that (1) the identity-based signature (IBS) scheme is existentially unforgeable under an adaptive chosen-message attack; (2) the public key encryption scheme E is secure against the chosen-ciphertext attack; (3) the decisional Diffie-Hellman problem is hard over (G, g, q); (4) the protocol P is a secure two-party PAKE protocol with explicit authentication; (5) H_1, H_2 are collision-resistant hash functions, then the protocol P' illustrated in Fig. 1 is a secure ID2S PAKE protocol according to Definition 1.

Proof. Given an adversary $\mathcal{A}$ attacking the protocol, we imagine a simulator $\mathcal{S}$ that runs the protocol for $\mathcal{A}$.

First of all, the simulator $\mathcal{S}$ initializes the system by generating $\mathsf{params} = \mathsf{params}^{\mathsf{P,IBS,E}} \bigcup \{(G, q, g), (H_1, H_2)\}$ and the secret $\mathsf{master\text{-}key}^{\mathsf{IBS}}$. Next, Client, Server, and Client ServerTriple sets are determined. Passwords for clients are chosen at random and split, and then stored at corresponding servers. Private keys for servers are computed using $\mathsf{master\text{-}key}^{\mathsf{IBS}}$.

The public information is provided to the adversary. Considering $(C, A, B) \in \mathsf{ClinetServerTriple}$, we assume that the adversary $\mathcal{A}$ chooses the server B to corrupt and the simulator $\mathcal{S}$ gives the adversary $\mathcal{A}$ the information held by the corrupted server B, including the private key of the server B, i.e., d_B, and one share of the password of the client C, $g^{\mathsf{pw}_{C,B}}$. After computing the appropriate answer to any oracle query, the simulator $\mathcal{S}$ provides the adversary $\mathcal{A}$ with the internal state of the corrupted server B involved in the query.

We view the adversary's queries to its Send oracles as queries to five different oracles as follows:

— $\mathsf{Send}(C, i, A, B)$ represents a request for instance C^i of client C to initiate the protocol. The output of this query is $\mathsf{msg} = \langle C, W_c \rangle$.

— $\mathsf{Send}(A, j, C, \mathsf{msg})$ represents sending message msg to instance A^j of the server A from C. The output of this query is $\mathsf{msg}_A = \langle A, W_a, pk_a, S_a \rangle$.

— $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$ represents sending the message $\mathsf{msg}_A|\mathsf{msg}_B$ to instance C^i of the client C. The output of this query is either $\mathsf{msg}_1 = \langle C, E_a\rangle|\mathsf{msg}_2 = \langle C, E_b\rangle$ or $\perp$.
— $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ represents sending message msg_1 to instance A^j of the server A from C. The output of this query is either $\mathsf{acc}_A = \mathsf{TRUE}$ or $\perp$.
— $\mathsf{Send}^P(A, j, B, M)$ represents sending message M to instance A^j of the server A, supposedly by the server B, in the two-party PAKE protocol P. The input and output of this query depends on the protocol P.

When $\mathcal{A}$ queries the Test oracle, the simulator $\mathcal{S}$ chooses a random bit b. When the adversary completes its execution and outputs a bit b', the simulator can tell whether the adversary succeeds by checking if (1) a single Test query was made regarding some fresh session key $\mathsf{sk}^i_{U,U'}$, and (2) $b' = b$. Success of the adversary is denoted by event Succ. For any experiment P', we denote $\mathsf{Adv}^{P'}_{\mathcal{A}} = 2 \cdot \mathsf{Pr}[\mathsf{Succ}] - 1$, where $\mathsf{Pr}[\cdot]$ denotes the probability of an event when the simulator interacts with the adversary in accordance with experiment P'.

We will use some terminology throughout the proof. A given message is called oracle-generated if it was output by the simulator in response to some oracle query. The message is said to be adversarially-generated otherwise. **An adversarially-generated message must not be the same as any oracle-generated message**.

We refer to the real execution of the experiment, as described above, as P_0. We introduce a sequence of transformations to the experiment P_0 and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment. This immediately yields a bound on the adversary's advantage in the original experiment.

Experiment P_1: In this experiment, the simulator interacts with the adversary as P_0 except that the adversary does not succeed, and the experiment is aborted, if any of the following occurs: 1) At any point during the experiment, an oracle-generated message (e.g., msg, msg_1, msg_2, msg_A, or msg_B) is repeated; 2) At any point during the experiment, a collision occurs in the hash function H_1 or H_2 (regardless of whether this is due to a direct action of the adversary, or whether this occurs during the course of the simulator's response to an oracle query).

It is immediate that events 1 occurs with only negligible probability, event 2 occurs with negligible probability assuming H_1, H_2 as collision-resistant hash functions. Put everything together, we are able to see that

Claim 1. If H_1 and H_2 are collision-resistant hash functions, $|\mathsf{Adv}^{P_0}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_1}_{\mathcal{A}}(k)|$ is negligible.

Experiment P_2: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_1 except that the adversary's queries to Execute oracles are handled differently: in any $\mathsf{Execute}(C, i, A, j, B, k)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $g^{\mathsf{pw}_1 h_1^{-1}}$ in the public key encryption E_a is replaced with a random element in G.

The difference between the current experiment and the previous one is bounded by the probability that an adversary breaks the semantic security of the public key encryption E. More precisely, we have

Claim 2. If the public key encryption scheme E is semantically secure, $|\mathsf{Adv}^{P_1}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_2}_{\mathcal{A}}(k)|$ is negligible.

Remark. If a public key encryption scheme is secure against the chosen-ciphertext attack (CCA), it is secure against the chosen-plaintext attack (CPA) (i.e., it is semantically secure).

Experiment P_3: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_2 except that: for any $\mathsf{Execute}(C, i, A, j, B, k)$ oracle, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the session keys $\mathsf{sk}_{C,A}$ and $\mathsf{sk}_{A,C}$ are replaced with a same random element in the group G.

The difference between the current experiment and the previous one is bounded by the probability to solve the decisional Diffie-Hellman (DDH) problem over (G, g, q). More precisely, we have

Claim 3. If the decisional Diffie-Hellman (DDH) problem is hard over (G, q, g), $|\mathsf{Adv}^{P_2}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_3}_{\mathcal{A}}(k)|$ is negligible.

If $|\mathsf{Adv}^{P_2}_{\mathcal{A}}(k) - \mathsf{Adv}^{P_3}_{\mathcal{A}}(k)|$ is non-negligible, we show that the simulator can use $\mathcal{A}$ as a subroutine to solve the DDH problem with non-negligible probability as follows.

Given a DDH problem (g^α, g^β, Z), where α, β are randomly chosen from Z^*_q and Z is either $g^{\alpha\beta}$ or a random element z from G, the simulator replaces W_c with g^α, and W_a with g^β, and the session keys $\mathsf{sk}_{C,A}$, $\mathsf{sk}_{A,C}$ with Z. When $Z = g^{\alpha\beta}$, the experiment is the same as the experiment P_2. When Z is a random element z in $\mathbb{G}$, the experiment is the same as the experiment P_3. If the adversary can distinguish the experiments P_2 and P_3 with non-negligible probability, the simulator can solve the DDH problem with non-negligible probability. Assuming that the DDH problem is hard, Claim 3 is true.

In experiment P_3, the adversary's probability of correctly guessing the bit b used by the Test oracle is exactly $1/2$ when the Test query is made to a fresh client instance C^i or a fresh server instance A^j invoked by an $\mathsf{Execute}(C, i, A, j, B, k)$ oracle, even if the adversary queried $\mathsf{corrupt}(B)$ (i.e., the adversary corrupted the server B). This is so because the session keys $\mathsf{sk}_{C,A}$ and $\mathsf{sk}_{A,C}$ for such instances in P_3 are chosen at random from G, and hence there is no way to distinguish whether the Test oracle outputs a random session key or the "actual" session key (which is just a random element, anyway). Therefore, all passive adversaries cannot win the game, even if they can query $\mathsf{Corrupt}(B)$ oracles.

The rest of the proof concentrates on the instances invoked by Send oracles.

Experiment P_4: In this experiment, we modify the simulator's responses to $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$ and $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ queries.

Before describing this change we introduce some terminology. For a query $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$, where $\mathsf{msg}_A|\mathsf{msg}_B$ is adversarially-generated, if $\mathsf{acc}^i_C = \mathsf{TRUE}$, then $\mathsf{msg}_A|\mathsf{msg}_B$ is said to be valid. Otherwise, it is said to be invalid. Similarly, for a query $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$, where msg_1 is adversarially-generated, if $\mathsf{acc}^j_A = \mathsf{TRUE}$, then msg_1 is said to be valid. Otherwise, msg_1 is said to be invalid. Informally, valid messages use correct passwords while invalid messages do not. Given this terminology, we

continue with our description of experiment P_4. When the adversary makes oracle query $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$, the simulator examines $\mathsf{msg}_A|\mathsf{msg}_B$. If the message is adversarially-generated and valid, the simulator halts and acc_C^i is assigned the special value ∇. In any other case, (i.e., $\mathsf{msg}_A|\mathsf{msg}_B$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment P_3. When the adversary makes oracle query $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$, the simulator examines msg_1. If it is adversarially-generated and valid, the simulator halts and acc_A^j is assigned the special value ∇. In any other case, (i.e., msg_1 is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in P_3.

Now, we change the definition of the adversary's success in P_4. At first, we define that a client instance C^i is fresh if the adversary has not queried $\mathsf{Corrupt}(C)$ and a server instance A^j is fresh if the adversary has not queried $\mathsf{Corrupt}(A)$. If the adversary ever queries $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance C^i with $\mathsf{acc}_C^i = \nabla$ or $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracle to a fresh server instance A^j with $\mathsf{acc}_A^j = \nabla$, the simulator halts and the adversary succeeds. Otherwise the adversary's success is determined as in experiment P_3.

The distribution on the adversary's view in experiments P_3 and P_4 are identical up to the point when the adversary queries $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance with $\mathsf{acc}_C^i = \nabla$ or $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracle to a fresh server instance with $\mathsf{acc}_A^j = \nabla$. If such a query is never made, the distributions on the view are identical. Therefore, we have

Claim 4. $\mathsf{Adv}_{\mathcal{A}}^{P_3}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_4}(k)$.

Experiment P_5: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_4 except that the adversary's queries to $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracles are handled differently: in any $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $g^{\mathsf{pw}_1 h_1^{-1}}$ in E_a is replaced with a random element in the group G.

As we prove Claims 2, we can prove

Claim 5. If the public key encryption scheme E is semantically secure, $|\mathsf{Adv}_{\mathcal{A}}^{P_4}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_5}(k)|$ is negligible.

In experiment P_5, msg_1 from Execute and Send oracles become independent of the password pw_C used by the client C in the view of the adversary $\mathcal{A}$, even if $\mathcal{A}$ may require $\mathsf{Corrupt}(B)$. In addition, although the adversary who has corrupted the server B is able to obtain g^{pw_2}, $g^{\mathsf{pw}_{C,B}}$, they are independent of the password pw_C in the view of the adversary because the reference msg_1 is independent of the password in the view of the adversary. In view of this, any off-line dictionary attack cannot succeed.

The adversary $\mathcal{A}$ succeeds only if one of the following occurs: (1) the adversary queries $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A|\mathsf{msg}_B)$ oracle to a fresh client instance C^i for adversarially-generated and valid $\mathsf{msg}_A|\mathsf{msg}_B$, that is, $\mathsf{acc}_C^i = \nabla$ (let Succ_1 denote this event); (2) the adversary queries $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracle to a fresh server instance A^j for adversarially-generated and valid msg_1, that is, $\mathsf{acc}_A^j = \nabla$ (let Succ_2 denote this event); (3) neither Succ_1 nor Succ_2 happens, the adversary wins the game by a Test query to a fresh instance C^i or a server instance A^j.

Claim 6. If the identity-based signature (IBS) scheme is existentially unforgeable under an adaptive chosen-message attack and the hash function H_1 is collision-resistant, $\mathsf{Pr}[\mathsf{Succ}_1]$ is negligible.

msg_A contains a signature S_a of the server A on $H_1(A, W_a, C, W_c, pk_a)$. If the adversary $\mathcal{A}$ is able to find W_a' ($\neq W_a$) or pk_a' ($\neq pk_a$) such that $H_1(A, W_a', C, W_c, pk_a') = H_1(A, W_a, C, W_c, pk)$ for given W_c, he may make $\mathsf{acc}_C^i = \nabla$ with the same S_a. However, this probability is negligible because we assume that the hash function H_1 is collision-resistant. If the adversary can forge a new signature S_a' of the server A on $H_1(A, W_a', C, W_c, pk_a')$ for chosen W_a' ($\neq W_a$) or pk_a' ($\neq pk_a$), he may also make $\mathsf{acc}_C^i = \nabla$. However, this probability is also negligible because we assume that the identity-based signature (IBS) scheme is existentially unforgeable under an adaptive chosen-message attack. Therefore, Claim 6 is true.

Claim 7. If (1) P is a secure two-party PAKE protocol with explicit authentication; (2) the public key encryption E is secure against the chosen-ciphertext attack; (3) H_2 is collision-resistant hash functions, then $\mathsf{Pr}[\mathsf{Succ}_2] \leq Q(k)/N + \varepsilon(k)$, where $Q(k)$ denotes the number of on-line attacks and $\varepsilon(k)$ is a negligible function.

To evaluate $\mathsf{Pr}[\mathsf{Succ}_2]$, we assume that the adversary $\mathcal{A}$ has corrupted the server B and consider three cases as follows.

Case 1. If the adversary $\mathcal{A}$ is able to find W_c' ($\neq W_c$) such that $H_2(C, W_c', A, W_a) = H_2(C, W_c, A, W_a)$ for given W_a, he may make $\mathsf{acc}_A^j = \nabla$ with the same E_a in msg_1. However, this probability is negligible because we assume that the hash function H_2 is collision-resistant.

Case 2. If the adversary $\mathcal{A}$ is able to replace the plaintext $g^{\mathsf{pw}_1 h_1^{-1}}$ in E_a with $g^{\mathsf{pw}_1 h_1^{*-1}}$ (like homomorphic encryption), where $h_1^* \neq h_1$, he may make $\mathsf{acc}_A^j = \nabla$ with E_a' transformed from E_a. However, this probability is negligible because we assume that the public key encryption E is secure against the chosen-ciphertext attack.

Case 3. The adversary $\mathcal{A}$ forges $\mathsf{msg}' = \langle C, W_c' \rangle$ and $\mathsf{msg}_1' = \langle C, E_a' \rangle$ where $E_a' = \mathsf{E}(g^{\mathsf{pw}_1^* h_1^{*-1}}, pk_a)$ and $h_1^* = H_2(C, W_c', A, W_a)$ and pw_1^* is chosen by the adversary. After receiving msg_1', the server A derives $\omega_a = g^{\mathsf{pw}_1^* - \mathsf{pw}_{C,A}} = g^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B} - \mathsf{pw}_C} = g^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B}} / g^{\mathsf{pw}_C}$ and then runs two-party PAKE protocol P with the server B (controlled by the adversary) on the basis of ω_a. In the view of the adversary, ω_a can be anyone of $\{g^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B}} / g^{\mathsf{pw}} | \mathsf{pw} \in \mathcal{D}\}$ even if he knows $\mathsf{pw}_1^* + \mathsf{pw}_{C,B}$. Without knowing ω_a, this probability of $\mathsf{acc}_A^P = \mathsf{TRUE}$ is less than $Q^P(k)/N + \varepsilon(k)$, where $Q^P(k)$ denotes the number of on-line attacks in the protocol P, because we assume that P is a secure two-party PAKE protocol with explicit authentication.

In experiment P_5, the adversary's probability of success when neither Succ_1 nor Succ_2 occurs is $1/2$. The preceding discussion implies that

$$\mathsf{Pr}_{\mathcal{A}}^{P_6}[\mathsf{Succ}] \leq Q(k)/N + \varepsilon(k) + 1/2 \cdot (1 - Q(k)/N - \varepsilon(k))$$

and thus the adversary's advantage in experiment P_5

$$\mathsf{Adv}_{\mathcal{A}}^{P_5}(k) \quad = \quad 2\mathsf{Pr}_{\mathcal{A}}^{P_5}[\mathsf{Succ}] - 1$$

$$\begin{aligned} &\leq 2Q(k)/N + 2\varepsilon(k) + 1 - Q(k)/N - \varepsilon(k) - 1 \\ &= Q(k)/N + \varepsilon(k) \end{aligned}$$

for some negligible function $\varepsilon(\cdot)$. The sequence of claims proved above show that

$$\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_5}(k) + \varepsilon(k) \leq Q(k)/N + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$. This completes the proof of the theorem.

4.2 Security of ID2S PAKE Protocol Based on IBE

Theorem 2. Assuming that (1) the identity-based encryption (IBE) scheme is secure against the chosen-ciphertext attack; (2) the public key encryption scheme E is secure against the chosen-ciphertext attack; (3) the decisional Diffie-Hellman problem is hard over (G, g, q); (4) the protocol P is a secure two-party PAKE protocol with explicit authentication; (5) H_1, H_2 are collision-resistant hash functions, then the protocol P' illustrated in Fig. 2 is a secure ID2S PAKE protocol according to Definition 1.

Proof. Given an adversary $\mathcal{A}$ attacking the protocol, a simulator $\mathcal{S}$ runs the protocol for $\mathcal{A}$. First of all, the simulator $\mathcal{S}$ initializes the system by generating $\mathsf{params} = \mathsf{params}^{\mathsf{P},\mathsf{IBE},\mathsf{E}} \bigcup \{(\mathbb{G}, \mathcal{G}, n), (G, q, g), (H_1, H_2)\}$ and the secret $\mathsf{master\text{-}key}^{\mathsf{IBE}}$. Next, Client, Server, and ClientServerTriple sets are determined. Passwords for clients are chosen at random and split, and then stored at corresponding servers. Private keys for servers are computed using $\mathsf{master\text{-}key}^{\mathsf{IBE}}$.

The public information is provided to the adversary. Considering $(C, A, B) \in \mathsf{ClinetServerTriple}$, we assume that the adversary $\mathcal{A}$ chooses the server B to corrupt and the simulator $\mathcal{S}$ gives the adversary $\mathcal{A}$ the information held by the corrupted server B, including the private key of the server B, i.e., d_B, and one share of the password of the client C, $\mathcal{G}^{\mathsf{pw}_{C,B}}$ and $g^{\mathsf{pw}^*_{C,B}}$. After computing the appropriate answer to any oracle query, the simulator $\mathcal{S}$ provides the adversary $\mathcal{A}$ with the internal state of the corrupted server B involved in the query.

We view the adversary's queries to its Send oracles as queries to four different oracles as follows:

— $\mathsf{Send}(C, i, A, B)$ represents a request for instance C^i of client C to initiate the protocol. The output of this query is $\mathsf{msg}_1 = \langle C, W_c, pk, E_a \rangle$ and $\mathsf{msg}_2 = \langle C, W_c, pk, E_b \rangle$.
— $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ represents sending message msg_1 to instance A^j of the server A. The output of this query is either $\mathsf{msg}_A = \langle A, W_a, E_1 \rangle$ or $\perp$.
— $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ represents sending the message $\mathsf{msg}_A | \mathsf{msg}_B$ to instance C^i of the client C. The output is either $\mathsf{acc}_C^i = \mathsf{TRUE}$ or $\perp$.
— $\mathsf{Send}^P(A, j, B, M)$ represents sending message M to instance A^j of the server A, supposedly by the server B, in the two-party PAKE protocol P. The input and output of this query depends on the protocol P.

We refer to the real execution of the experiment, as described above, as P_0.

Experiment P_1: In this experiment, the simulator interacts with the adversary as P_0 except that the adversary does not succeed, and the experiment is aborted, if any of the following occurs:

1) At any point during the experiment, an oracle-generated message is repeated.
2) At any point during the experiment, a collision occurs in the hash function H_1 or H_2

Claim 1. If H_1 and H_2 are collision-resistant hash functions, $|\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_1}(k)|$ is negligible.

Experiment P_2: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in P_1 except that the adversary's queries to Execute oracles are handled differently: in any $\mathsf{Execute}(C, i, A, j, B, k)$, where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, (1) the plaintext $\mathcal{G}^{\mathsf{pw}_1 h^{-1}}$ in E_a is replaced with a random element in $\mathbb{G}$; (2) the plaintext $g^{\mathsf{pw}^*_{C,A} h_a^{-1}}$ in E'_a is replaced by a random element in G; (3) the session keys $\mathsf{sk}_{C,A}$ and $\mathsf{sk}_{A,C}$ are replaced with a same random element in G.

Claim 2. If (1) the identity-based encryption (IBE) scheme is secure against the chosen-ciphertext attack; (2) the public key encryption scheme E is secure against the chosen-ciphertext attack; (3) the decisional Diffie-Hellman problem is hard over (G, g, q), $|\mathsf{Adv}_{\mathcal{A}}^{P_1}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_2}(k)|$ is negligible.

Experiment P_3: In this experiment, we modify the simulator's responses to $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ and $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ queries. If the adversary ever queries $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracle to a fresh server instance A^j with a adversarially-generated and valid msg_1 (denoted as $\mathsf{acc}_A^j = \nabla$) or $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ oracle to a fresh client instance C^i with a adversarially-generated and valid $\mathsf{msg}_A | \mathsf{msg}_B$ (denoted as $\mathsf{acc}_C^i = \nabla$), the simulator halts and the adversary succeeds. Otherwise the adversary's success is determined as in experiment P_2.

Claim 3. $\mathsf{Adv}_{\mathcal{A}}^{P_2}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_3}(k)$.

Experiment P_4: In this experiment, the simulator interacts with the adversary $\mathcal{A}$ as in experiment P_3 except that the adversary's queries to $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ and $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ oracles are handled differently: in any $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ or $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ oracles where the adversary $\mathcal{A}$ has not queried $\mathsf{corrupt}(A)$, but may have queried $\mathsf{corrupt}(B)$, the plaintext $\mathcal{G}^{\mathsf{pw}_1 h^{-1}}$ in E_a is replaced with a random element in $\mathbb{G}$ and the plaintext $g^{\mathsf{pw}^*_{C,A} h_a^{-1}}$ in E'_a (if any) is replaced with a random element in G.

Claim 4. If both the IBE scheme and the public key encryption scheme E are semantically secure, $|\mathsf{Adv}_{\mathcal{A}}^{P_3}(k) - \mathsf{Adv}_{\mathcal{A}}^{P_4}(k)|$ is negligible.

The adversary $\mathcal{A}$ succeeds only if one of the following occurs: (1) the adversary queries $\mathsf{Send}(A, j, C, \mathsf{msg}_1)$ oracle to a fresh server instance A^j with $\mathsf{acc}_A^j = \nabla$ (let Succ_1 denote this event); (2) the adversary queries $\mathsf{Send}(C, i, A, B, \mathsf{msg}_A | \mathsf{msg}_B)$ oracle to a fresh client instance C^i with $\mathsf{acc}_C^i = \nabla$ (let Succ_2 denote this event); (3) neither Succ_1 nor Succ_2 happens, the adversary wins the game by a Test query to a fresh instance C^i or a server instance A^j.

To evaluate $\mathsf{Pr}[\mathsf{Succ}_1]$ and $\mathsf{Pr}[\mathsf{Succ}_2]$, we assume that the adversary $\mathcal{A}$ has corrupted the server B and consider two as follows.

Case 1. The adversary $\mathcal{A}$ forges $\mathsf{msg}'_1 = \langle C, W'_c, pk', E'_a \rangle$ where $E'_a = \mathsf{IBE}(\mathcal{G}^{\mathsf{pw}_1^* h^{*-1}}, A), h^* = H_1(C, W'_c, pk')$, $W'_c, pk', \mathsf{pw}_1^*$ are chosen by the adversary, and sends msg'_1

to the server A. After receiving msg'_1, the server A derives $\omega_a = \mathcal{G}^{\mathsf{pw}_1^* - \mathsf{pw}_{C,A}} = \mathcal{G}^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B} - \mathsf{pw}_C} = \mathcal{G}^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B}} / \mathcal{G}^{\mathsf{pw}_C}$ and then runs two-party PAKE protocol P with the server B (controlled by the adversary) on the basis of ω_a. In the view of the adversary, ω_a can be anyone of $\{\mathcal{G}^{\mathsf{pw}_1^* + \mathsf{pw}_{C,B}} / \mathcal{G}^{\mathsf{pw}} | \mathsf{pw} \in \mathcal{D}\}$. If P is a secure two-party PAKE protocol with explicit authentication, $\mathsf{Pr}[\mathsf{Succ}_1] \leq Q^P(k)/N + \varepsilon(k)$.

Case 2. Given C, W_c, for $X = A$ and B, $x = a$ and b, the adversary $\mathcal{A}$ forges $\mathsf{msg}'_X = \langle X, W'_x, E'_x \rangle$, where $E'_x = \mathsf{E}(g^{\mathsf{pw}_x^* h_x^{*-1}}, pk)$, $h_x^* = H_2(X, W'_x, C, W_c)$, and W'_x, pw_x^* are chosen by the adversary. Then $\mathcal{A}$ sends $\mathsf{msg}'_A | \mathsf{msg}'_B$ to the client. In this case, the event Succ_2 occurs if and only if $\mathsf{pw}_a^* + \mathsf{pw}_b^* = \mathsf{pw}_C$. Therefore, $\mathsf{Pr}[\mathsf{Succ}_2] \leq Q^C(k)/N$, where $Q^C(k)$ denotes the number of on-line attacks to the client instance C^i.

The above discussion shows that

Claim 5. If (1) P is a secure two-party PAKE protocol with explicit authentication; (2) the IBE scheme and the public key encryption scheme are secure against the chosen-ciphertext attack; then $\mathsf{Pr}[\mathsf{Succ}_1 \vee \mathsf{Succ}_2] \leq Q(k)/N + \varepsilon(k)$, where $Q(k)$ denotes the number of on-line attacks.

In P_4, the adversary's probability of success when neither Succ_1 nor Succ_2 occurs is $1/2$. This implies that

$$\mathsf{Pr}_{\mathcal{A}}^{P_4}[\mathsf{Succ}] \leq Q(k)/N + \varepsilon(k) + 1/2 \cdot (1 - Q(k)/N - \varepsilon(k))$$

and thus the adversary's advantage in experiment P_0

$$\mathsf{Adv}_{\mathcal{A}}^{P_0}(k) \leq \mathsf{Adv}_{\mathcal{A}}^{P_4}(k) + \varepsilon(k) \leq Q(k)/N + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$. This completes the proof.

5 Performance Analysis

The efficiency of the compiled protocols using our compilers depends on performance of the underlying protocols.

In our IBS-based protocol, if we use the KOY two-party PAKE protocol [22], the Paterson et al.'s IBS scheme [25] and the Cramer-Shoup public key encryption scheme [13] as cryptographic building blocks, the performance of our IBS-based protocol can be shown in TABLE 1. In our IBE-based protocol, if we use the KOY two-party PAKE protocol [22], the Waters IBE scheme [28] and the Cramer-Shoup public key encryption scheme [13] as cryptographic building blocks, the performance of our IBE-based protocol can also be shown in TABLE 1. In addition, we compare our protocols with the Katz et al. two-server PAKE protocol [23] (secure against active adversary) in TABLE 1. In TABLE 1, Exp., exp. Sign. and Pair for computation represent the computation complexities of a modular exponentiation over an elliptic curve, a modular exponentiation over Z_p, a signature generation and a pairing, respectively, and Exp., exp. and Sign. in communication denote the size of the modulus and the size of the signature, and KOY stands for the computation or communication complexity of the KOY protocol.

In TABLE 1, different operations are computed in different protocols. For example, some modular exponentiations in our protocols are over an elliptic curve group, while the modular exponentiations in the Katz et al.'s protocol are over Z_p only. Our protocols need to compute pairings while the Katz et al.'s protocol does not. In order to further compare their performance, we implement our two protocols.

To realize the modular exponentiation $\mathcal{G}^x$ over an elliptic curve group $\mathbb{G}$ and the pairing map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ in our protocols, we build our implementation on top of the PBC pairing-based cryptography library[1], whereas the multiplicative group over the prime integer p is based on the GNU MP library[2]. Moreover, the elliptic curve we use is the A512 ECC in which the first two groups are the same, i.e., a symmetric pairing. Another library mbed TLS[3] is adopted due to the invocations of AES and SHA-512 for the one-time signature in KOY. All the experiments were conducted in Ubuntu 14.04 running on a computer equipped with an Intel i7-4770HQ CPU and 16 GBytes of memory. When implementing our protocols, we also performed optimization when applicable. For example, we compute the Waters' hash function by parallel computation.

The execution time of our two protocols compared with the Katz et al.'s protocol can be shown in TABLE 2. From TABLE 2, we can see that the client performance in Katz et al.'s protocol is better than our protocols, but the execution times for client in the three protocols are all less than 10 ms. The server performance in our protocols is better than the Katz et al.'s protocol, saving from 22% to 66% of computation. When the servers provide services to a great number of clients concurrently, the server performance is critical to the performance of the whole protocol. For example, assume that Servers A and B provide services to 100 clients concurrently and there is no communication delay, the longest waiting time with respect to a client for our IBE-based protocol is around 7.08+208+176=391.08 ms while the Katz et al.'s protocol takes about 1.26+531+531=1,063.26 ms. The difference is 672.18 ms.

In terms of communication complexity, the size of a group element over elliptic curve (denoted as Exp.) in our protocols can be 512 bits, while the size of a group element over Z_p in the Katz et al.'s protocol [23] has to be 1024 bits. From TABLE 1, we can see that the communication complexity of our protocols is about a half of the Katz et al.'s protocol [23].

6 Conclusion

In this paper, we present two efficient compilers to transform any two-party PAKE protocol to an ID2S PAKE protocol with identity-based cryptography. In addition, we have provided a rigorous proof of security for our compilers without random oracle. Our compilers are in particular suitable for the applications of password-based authentication where an identity-based system has already established. Our future work is to construct an identity-based multiple-server PAKE protocol with any two-party PAKE protocol.

References

[1] M. Abdalla, P. A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *Proc. PKC'05*, pages 65-84, 2005.

1. https://crypto.stanford.edu/pbc/download.html
2. https://gmplib.org/
3. https://tls.mbed.org/

TABLE 1
Performance comparison of Katz et al. protocol and our protocols

	Katz et al. Protocol [23]	Our IBS-based Protocol	Our IBE-based Protocol
Public Keys	Client: None Sever A: Public Key pk_A Sever B: Public Key pk_B	Client: None Server A: A Server B: B	Client: None Server A: A Server B: B
Private Keys	Client: pw_C Sever A: $\mathsf{pw}_{C,A}$, Private Key sk_A Sever B: $g^{\mathsf{pw}_{C,B}}$, Private Key sk_B	Client: pw_C Server A: $g^{\mathsf{pw}_{C,A}}, d_A$ Server B: $g^{\mathsf{pw}_{C,B}}, d_B$	Client: pw_C Server A: $\mathcal{G}^{\mathsf{pw}_{C,A}}, g^{\mathsf{pw}^*_{C,A}}, d_A$ Server B: $\mathcal{G}^{\mathsf{pw}_{C,B}}, g^{\mathsf{pw}^*_{C,B}}, d_B$
Computation Complexity	Client: 21(exp.)+1(Sign) Server: about 6(KOY)	Client: 23(Exp.)+6(Pair) Server: about 1(KOY)+9(Exp)	Client: 23(Exp.) Server: about 1(KOY)+2(Pair)+9(Exp.)
Communication Complexity	Client/Server: 27(exp.)+1(Sign) Server/Server: about 2(KOY)	Client/Server: 22(Exp.) Server/Server: about 1(KOY)	Client/Server: 24(Exp.) Server/Server: about 1(KOY)

TABLE 2
Execution Time of Protocols (in milliseconds)

	Katz et al. Protocol [23]	Our IBS-based Protocol	Our IBE-based Protocol
Client	1.26	5.26	7.08
Server A	5.31	4.14	2.08
Server B	5.31	3.82	1.76

[2] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *Proc. CT-RSA 2005*, pages 191-208, 2005.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proc. Eurocrypt'00*, pages 139-155, 2000.

[4] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocol secure against dictionary attack. In *Proc. 1992 IEEE Symposium on Research in Security and Privacy*, pages 72-84, 1992.

[5] J. Bender, M. Fischlin, and D. Kugler. Security analysis of the PACE key-agreement protocol. In *Proc. ISC'09*, pages 33-48, 2009.

[6] J. Bender, M. Fischlin, and D. Kugler. The PACE|CA protocol for machine readable travel documents. In *INTRUST'13*, pages 17-35, 2013.

[7] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *Proc. Crypto'01*, pages 213-229, 2001.

[8] V. Boyko, P. Mackenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Proc. Eurocrypt'00*, pages 156-171, 2000.

[9] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A new two-server approach for authentication with short secrets. In*Proc. 12th USENIX Security Symp.*, pages 201-213, 2003.

[10] E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In *Proc. CCS'03*, pages 241-250, 2003.

[11] E. Bresson, O. Chevassut, and D. Pointcheval. New security results on encrypted key exchange. In *Proc. PKC'04*, pages 145-158, 2004.

[12] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32 (9): 33-38, 1994.

[13] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proc. Crypto'98*, pages 13-25, 1998.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 32(2): 644-654, 1976.

[15] W. Ford and B. S. Kaliski. Server-assisted generation of a strong secret from a password. In *Proc. 5th IEEE Intl. Workshop on Enterprise Security*, 2000.

[16] O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *Proc. Crypto'01*, pages 408-432, 2001.

[17] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly-chosen secret from guessing attacks. *IEEE J. on Selected Areas in Communications*, 11(5):648-656, 1993.

[18] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security*, 2(3):230-268, 1999.

[19] D. Jablon. Password authentication using multiple servers. In *Proc. CT-RSA'01*, pages 344-360, 2001.

[20] S. Jiang and G. Gong. Password based key exchange with mutual authentication. In *Proc. SAC'04*, pages 267-279, 2004.

[21] H. Jin, D. S. Wong, and Y. Xu. An efficient password-only two-server authenticated key exchange system. In *Proc. ICICS'07*, pages 44-56,2007

[22] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Proc. Eurocrypt'01*, pages 457-494, 2001.

[23] J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Proc. ACNS'05*, pages 1-16, 2005.

[24] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. *J. Cryptology*, 19(1): 27-66, 2006.

[25] K. G. Paterson and J. C.N. Schuldt. Efficient identity-based signatures secure in the standard model. In *ACISP'06*, pages 207-222, 2006.

[26] M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. *J. Computer and System Sciences*, 72(6): 978-1001 (2006).

[27] M. Szydlo and B. Kaliski. Proofs for two-server password authentication. In *Proc. CT-RSA'05*, pages 227-244, 2005.

[28] B. Waters. Efficient identity-based encryption without random oracles. In *Proc. Eurocrypt'05*, pages 114-127, 2005.

[29] Y. Yang, F. Bao, R. H. Deng. A new architecture for authentication and key exchange using password for federated enterprise. In *Proc. SEC'05*, pages 95-111, 2005.

[30] Y. Yang, R. H. Deng, and F. Bao. A practical password-based two-server authentication and key exchange system. *IEEE Trans. Dependable and Secure Computing*, 3(2), 105-114, 2006.

[31] Y. Yang, R. H. Deng, and F. Bao. Fortifying password authentication in integrated healthcare delivery systems. In *Proc. ASI-ACCS'06*, pages 255-265,2006.

[32] X. Yi, R. Tso and E. Okamoto. ID-based group password-authenticated key exchange. In *Proc. IWSEC'09*, pages 192-211, 2009.

[33] X. Yi, R. Tso and E. Okamoto. Identity-based password-authenticated key exchange for client/server model. In *SECRYPT'12*, pages 45-54, 2012.

[34] X. Yi, S. Ling, and H. Wang. Efficient two-server password-only authenticated key exchange. *IEEE Trans. Parallel Distrib. Syst.* 24(9): 1773-1782, 2013.

[35] X. Yi, F. Hao and E. Bertino. ID-based two-server password-authenticated key exchange. In *ESORICS'14*, pages 257-276, 2014.